# Comparing Application Layer Protocols for the Internet of Things via Experimentation

Stefan Mijovic, Erion Shehu, Chiara Buratti
DEI, University of Bologna
Viale del Risorgimento 2, 40136 Bologna, Italy
stafan.mijovic@unibo.it, erion.shehu2@studio.unibo.it, c.buratti@unibo.it

*Abstract*—**In this paper we compare the performance of three application layer protocols, that are Constrained Application Protocol (CoAP), WebSocket and Message Queuing Telemetry Transport (MQTT), in an Internet of Things (IoT) scenario. The three protocols have been implemented on the same low cost and low complexity hardware platform, suitable for IoT applications. The performance, in terms of protocol efficiency, strictly related to the overhead, and average Round Trip Time (RTT), were experimentally evaluated. In the considered scenario an IoT device was transmitting data to a server and waiting for replies. IEEE 802.11.b/g/n air interface was used for the communication between the IoT device and an Access Point (AP), connected to the final server. Two different settings have been considered: a local area network (LAN) configuration, where the AP and the server were in the same LAN; and a more realistic IoT configuration, where the AP was connected to a remote server via the Internet. Furthermore, in the IoT configuration, two types of Internet connection are considered: a connection established through a home router and another via cellular network. Results show that CoAP achieves the highest protocol efficiency and the lowest average RTT, closely followed by WebSocket. The performance of MQTT protocol strongly depend on the Quality of Service (QoS) profile. Changing the environment, from a LAN network to a realistic IoT scenario, does not significantly impact the protocol efficiency, but has a considerable influence on the average RTT, which increases by a factor of 2 or 3, depending on the protocol. Finally, we give some insights on the impact of routing through a cellular network on the system performance.**

## I. Introduction

The number of sensors pervading our everyday life, e.g., in smartphones, cars, and buildings, is constantly increasing [1]. These sensors, which are typically embedded into resource-constrained devices such as sensor nodes or smartphones, allow measuring the state of the entities they observe or are attached to. If this information is accessible via the Internet, they contribute to the Internet of Things (IoT), where real-world objects have virtual representations. Given the important position of the IoT within the 5G context, the standardisation process to have a complete and efficient communication protocol stack for these constrained devices is in full swing.

An IoT system, as every other communication system, can be mapped on the Open System Interconnection (OSI) model. The application layer is the seventh and final layer of the OSI model for computer networks. It provides services for an application program to ensure that effective communication with another application program in a network is possible. Historically different application fields had their ”de facto” standard application layer, such as HyperText Transfer Protocol (HTTP) for World Wide Web, however this is changing nowadays and many new protocols are emerging.

IoT communication protocols are a subject of many studies in research community. The work presented in [2] provides a comprehensive survey of the enabling technologies, protocols, and architecture for an urban IoT. Furthermore, [2] presents and discusses the technical solutions and best-practice guidelines obtained from a proof-of-concept deployment of an IoT system. On the other hand [3] justifies the necessity of a standardised communication protocol stack for the IoT and proposes such a stack which is based on various standards from IEEE 802.15.4 and IETF working group.

Application layer protocols for the IoT were investigated in [4], where the authors provide a quantitative performance analysis of the Constrained Application Protocol (CoAP) and Message Queuing Telemetry Transport (MQTT) over various conditions of offered traffic, packet loss probability and delay. In this study IoT devices are emulated on virtual machines. Another analysis of application layer protocols for IoT is presented in [5] where a cellular network-based approach is considered. The authors use cellular network emulator for EDGE, UMTS and LTE to analyse the protocols without the presence of delays caused by the Internet or by other users allocating resources of the cellular network. Similar topic is analysed in [6], where MQTT and CoAP are experimentally compared through an emulated lossy network.

With respect to the literature, in this work we take a fully experimental approach. The considered application layer protocols have been implemented on a typical IoT hardware platform thus allowing for a realistic performance estimation. Being implemented on the same platform, the comparison between protocols is fair. Considered performance metrics were protocol efficiency and average Round Trip Time (RTT).

The rest of the paper is organised as follows. Section II introduces the three application layer protocols under investigation. Section III describes the experimental setup and hardware and software used. Section IV reports the results of the experiments, while Section V concludes the paper.

## II. Application Layer Protocols

In this paper we focus on the application layer of the OSI model and we investigate three different protocols: CoAP [7], WebSocket [8] and MQTT [9]. These three protocols originate from different application fields[1] but have certain properties which make them suitable for IoT applications.

### A. CoAP

CoAP is a specialised web transfer protocol for use with constrained nodes and constrained networks in the IoT. CoAP protocol stack resources are identified using hierarchical Uniform Resource Identifier (URI) schema. It is designed to easily translate to HTTP for simplified integration with the web, while also meeting specialised requirements such as multicast support, very low overhead, and simplicity. The CoAP messaging model is based on the exchange of messages over User Datagram Protocol (UDP) between endpoints.

### B. WebSocket

WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. WebSocket is a protocol providing full-duplex communication channels over a single Transmission Control Protocol (TCP) connection. This is made possible by providing a standardised way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. The connection gets established by ”upgrading” from HTTP to HTTP 1.1.

### C. MQTT

MQTT uses publish/subscribe messaging transport protocol. The publish/subscribe pattern (pub/sub) is an alternative to the traditional client-server model, where a client communicates directly with an endpoint. However, pub/sub decouples a client, who is sending a particular message (called publisher) from another client (or more clients), who is receiving the message (called subscriber). This means that the publisher and the subscriber do not know about the existence of one-another. There is a third component, called broker, which is known by both the publisher and subscriber, which filters all incoming messages and distributes them accordingly. MQTT uses different models for message exchange known as Quality of Service (QoS) profiles. QoS profiles considered in this paper are:

- QoS0: the publisher sends the data to the broker and does not wait for Acknowledgement (ACK) from the broker. In this profile there are no retransmissions and if the data sent is not received, it gets lost;
- QoS1: to avoid the previously mentioned problem, after publishing the data, the publisher waits for an ACK (called PUBACK) from the broker. If it does not receive the ACK after a predefined amount of time, it retransmits the data. This profile improves reliability but it increases the overhead.
- QoS2: the publisher sends a data to the broker and waits for PUBREC (PUBlish RECeived) message back from it. When it receives the PUBREC it discards the reference to the published data and sends a PUBREL (PUBlish RELeased) to the broker. The broker follows the same procedure. When the flow is completed both parties can be sure that the message has been delivered. This profile is characterised by a very large delay and for this reason it is not considered in our measurements.

MQTT relies on TCP as transport layer protocol.

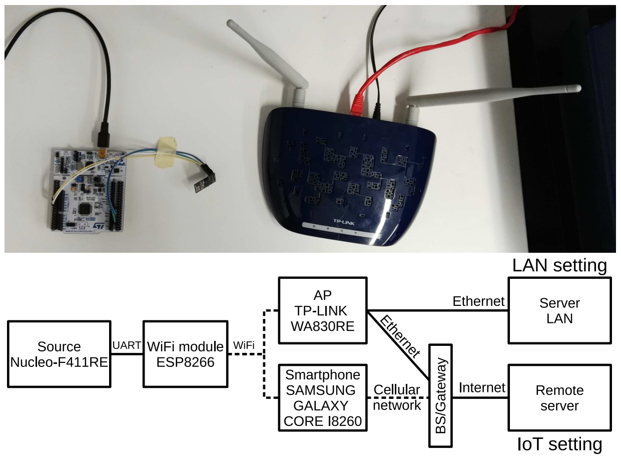

Fig. 1. Experimental setup.

## III. Experimental Setup and Metrics

In order to have a realistic estimation of the performance of an IoT system, suitable hardware must be chosen, i.e., affordable and low-complexity devices. For the experiments we used a popular STMicroelectronics Nucleo-F411RE development board [10]. Its key features are summarised below:

- STM32F411RE microcontroller, based on a high-performance ARM Cortex-M4 32-bit RISC core operating at a frequency of up to $100\,\mathrm{MHz}$[2];
- $512\,\mathrm{kB}$ of flash memory and $128\,\mathrm{kB}$ of RAM;
- Arduino and Morpho headers for easy interfacing with other hardware components.

The connectivity is provided to the Nucleo board via ESP8266 module [11], which is a very popular and extremely cheap WiFi module capable of running a full TCP/IP stack. Communication between the Nucleo board and the ESP8266 module is established via AT commands over Universal Asynchronous Receiver/Transmitter (UART) interface. The ESP8266 module is further connected via WiFi to an Access Point (AP) and from there to the server. We consider the following experimental setups (see Figure 1):

[1]E.g., CoAP was designed with simple devices in mind, WebSocket is intended to be integrated into web browsers and web servers, while MQTT was originally supposed to be used over a satellite link.

[2]In our experiments the clock frequency was $16\,\mathrm{MHz}$

- Local Area Network (LAN) configuration: the AP and the server are in the same LAN, thus having a reliable low-latency link;
- IoT configuration: the AP and the server are not in the same LAN, but they are connected via Internet; in particular the remote server was located at the University of Bologna, while the AP was located in a private home. In this case performance was affected by the routing process and network congestion. To gain some intuition about the impact of the previously mentioned aspects on the system performance, we did experiments with two types of Internet connection:
  - via commercial Internet Service Provide (ISP): in this case ESP8266 module is connected to a home router;
  - via cellular network: ESP8266 module is connected to a cellular phone in tethering mode.

Regarding the software, the application layer was implemented on Nucleo board using ARM-mbed libraries, while the lower layers of the stack were running on the ESP8266 module. On the other hand, there are several implementations of the considered protocols for a regular computers. In our experiments the server used Libcoap, Tornado and Mosquitto for CoAP, WebSocket and MQTT, respectively.

In all the reported experiments, a data packet, emulating sensor/actuator data, was generated by the Nucleo board and transmitted through UART interface to the ESP8266 module which in turn forwarded the packet through WiFi to the AP. From then on, data was routed to the final server. The server replied with the same length data and via the same path to the Nucleo board. For Websocket, being full duplex, and CoAP, being designed with query-reply applications in mind, the server response was already embedded in the application. In the case of MQTT, this was achieved by making the Nucleo board both a subscriber and a publisher on the same topic. Numerical results are obtained by averaging over 10000 generated data packets.

Two metrics were considered for performance evaluation: the protocol efficiency and the average RTT.

We define the protocol efficiency, $\eta$, as the ratio between the number of useful information bytes, i.e., application layer payload, and the total number of bytes exchanged at application and transport layers. Protocol efficiency was computed starting from traffic logs, obtained at the server side via open-source packet analyser Wireshark.

The average RTT is defined as the average period of time required for a data packet to make a trip from the application layer of the source, the Nucleo board, to the destination, the server, and back. The RTT includes:

- the time needed for handshaking procedure, where applicable;
- the time needed for packet framing;
- the time needed for the transmission of the packet from the Nucleo Board to the ESP8266 module via UART;
- the time needed for the transmission of the packet from ESP8266 module to the AP via WiFi;
- the time needed for routing the packet from the AP to the server through the Internet;
- the time needed for the same travel in the opposite direction.

The RTT of a packet is computed at the IoT device side, i.e., Nucleo board, by means of timestamps. One timestamp is taken at the packet generation instant at the application layer, while the second timestamp is taken at the instant of reception of the server response, again at the application layer. Since both timestamps are taken at the same device, there is no time reference problem and RTT at the application layer can be accurately estimated.

## IV. Experimental Results

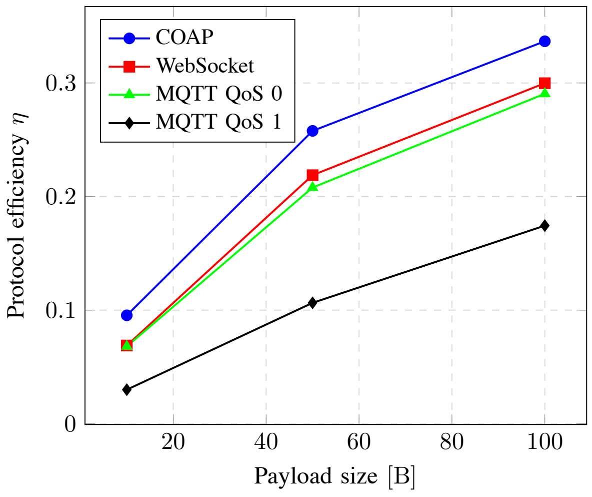

Fig. 2. Protocol efficiency LAN scenario.

First we report the experimental results obtained in LAN. Figure 2 shows the protocol efficiency as a function of the payload size in bytes for the three protocols in a LAN network. The highest efficiency is achieved by CoAP, followed by WebSocket and MQTT QoS0. The reason for this is that CoAP uses UDP as the transport layer protocol. With respect to TCP, UDP has less header and lacks transport layer ACKs, which makes it very efficient. As an example, when transmitting 100 bytes of payload, the protocol efficiency is around $\frac{1}{3}$ which means that one out of three exchanged bytes is an information byte. On the other hand, MQTT QoS1 achieves the lowest efficiency since it not only relies on TCP but also employs application layer ACKs. Figure 3 presents the average RTT as a function of the payload size. We can observe that CoAP, WebSocket and MQTT with QoS0 achieve very similar RTT, while MQTT with QoS1 has the highest RTT due to the presence of both transport and application layer ACKs.

Figures 4 and 5 report the experimental results in the IoT setting. In Figure 4 the protocol efficiency is shown as a

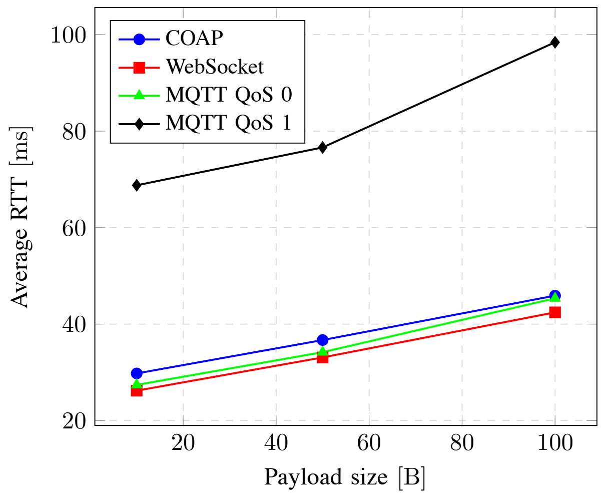

Fig. 3. Average RTT LAN scenario.

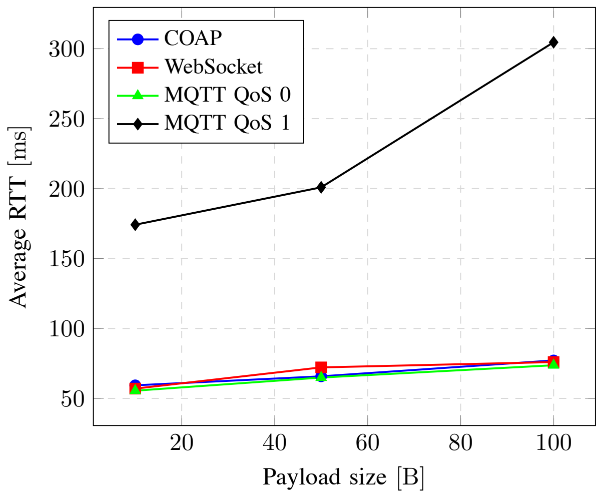

Fig. 5. Average RTT IoT scenario - ISP.

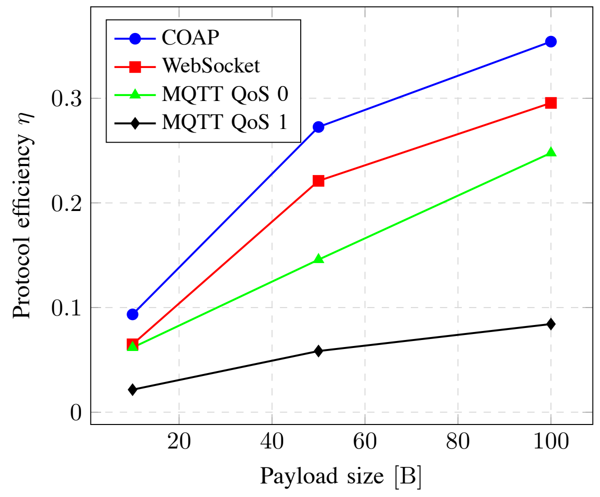

Fig. 4. Protocol efficiency IoT scenario - ISP.

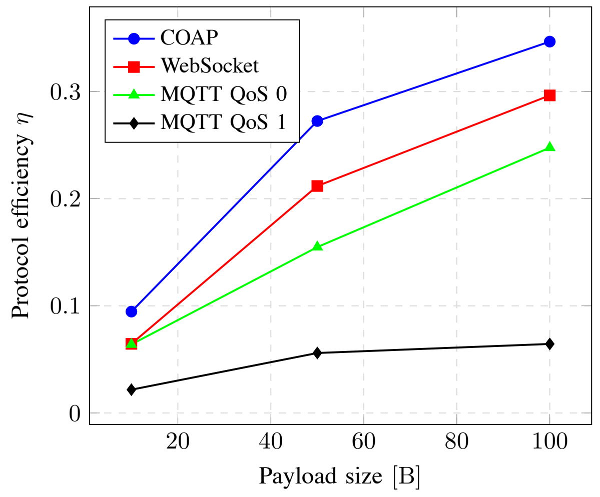

Fig. 6. Protocol efficiency IoT scenario - cellular network routing.

function of the payload size. We can see that both the trends and the values of the curves closely follow those presented in Figure 2, bringing to the conclusion that the protocol efficiency is not significantly affected by the network setting. On the other hand, Figure 5 presents the average RTT as a function of the payload size. Comparison with Figure 3 shows an increase in RTT by a factor of 2 to 3, depending on the protocol.

Finally, we report the results of the experiments in the IoT configuration where the role of the AP is played by a smartphone in tethering mode. In this case, the network through which the packets are routed, i.e., a cellular network, is much more volatile with respect to the previous case since it depends on random parameters, such as the network traffic within a cell, as well as various wireless propagation aspects. Since the randomness of QoS within a cellular network is time-dependent and the experiments were not performed contemporaneously, the results of this set of measurements must not be taken as a fair comparison among protocols, but merely as an insight on the effects of a volatile network on the performance.

Figure 6 presents the protocol efficiency as a function of the payload size. Comparing this figure with Figures 2 and 4 does not reveal any significant difference among them. This reinforces the previously stated hypothesis that the protocol efficiency is not significantly affected by the network settings. The same cannot be stated for the average RTT. Figure 7 presents the average RTT as a function of the payload size. While some protocols achieve similar performance as in

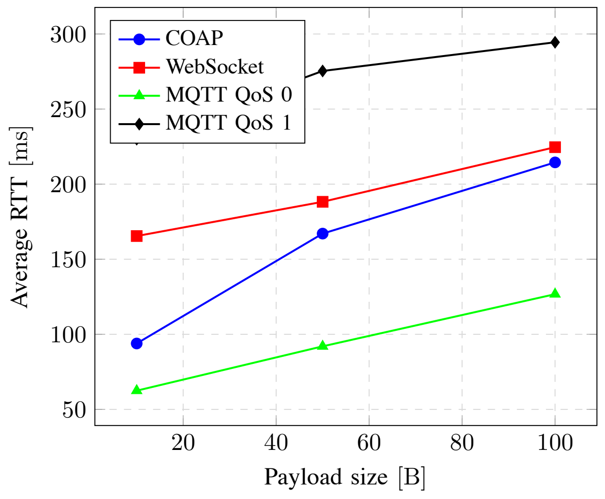

Fig. 7. Average RTT IoT scenario - cellular network routing.

IoT configuration with an AP (MQTT QoS0 and QoS1, see Figure 5), performance of others (WebSocket and CoAP) are greatly affected. However, this fact does not allow us to conclude that WebSocket and CoAP are less resilient to network volatility than MQTT since at another time instance reverse results could be achieved. In order to properly address this issue, a much longer experimental campaign should be conducted.

## V. Conclusions

The topic under investigation of this paper was the performance evaluation of three application layer protocols in an IoT scenario: CoAP, WebSocket and MQTT. The experimental approach was taken and the considered protocols were implemented on an affordable and simple devices intended for IoT applications, thus allowing for a fair comparison and realistic performance evaluation. Two scenarios were considered, a LAN and a more realistic IoT network and the performance was evaluated in terms of the protocol efficiency and the average RTT. The results of the experiments show that the protocol efficiency does not change by changing the network setting, from LAN to an IoT network. On the other hand, the average RTT is from two to three times higher in the second case. Furthermore, the case in which the traffic was routed through a cellular network shows that the average RTT can be greatly affected by the network state, i.e., cell traffic load, propagation aspects, etc. The results also showed that the CoAP achieves the highest efficiency with respect to the other protocols while MQTT with QoS1 has both the lowest efficiency and the highest RTT due to the presence of both transport and application layer ACKs.

## References

[1] "CISCO whitepaper: How the Next Evolution of the Internet Is Changing Everything," http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for Smart Cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[3] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler, "Standardized Protocol Stack for the Internet of (Important) Things," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[4] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G. E. Corazza, "Internet of Things application layer protocol analysis over error and delay prone links," in *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, Sept 2014, pp. 398–404.

[5] L. Durkop, B. Czybik, and J. Jasperneite, "Performance evaluation of M2M protocols over cellular networks in a lab environment," in *18th International Conference on Intelligence in Next Generation Networks (ICIN), 2015*, Feb 2015, pp. 70–75.

[6] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014*, April 2014, pp. 1–6.

[7] "The Constrained Application Protocol (CoAP)," https://tools.ietf.org/html/rfc7252.

[8] "The WebSocket Protocol," https://tools.ietf.org/html/rfc6455.

[9] "MQTT, the lightweight publish/subscribe messaging transport protocol," http://mqtt.org/.

[10] "STM32 Nucleo-64 development board with STM32F411RE MCU," http://www2.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-nucleo/nucleo-f411re.html.

[11] "Espressif ESP8266, a highly integrated WiFi module." http://espressif.com/en/products/hardware/esp8266ex/overview.